Gremlin Graphics Modes

Table of Contents

[Available Resolutions 2](#_Toc307084987)

[Video Modes 2](#_Toc307084988)

[RGB 2](#_Toc307084989)

[256 Color 2](#_Toc307084990)

[16 Color 2](#_Toc307084991)

[Display Layers 3](#_Toc307084992)

[Character Layout 3](#_Toc307084993)

[FrameBuffer Layout 3](#_Toc307084994)

[VRAM 4](#_Toc307084995)

[Kinds of VRAM 4](#_Toc307084996)

[Character Maps 4](#_Toc307084997)

[Character Cells 4](#_Toc307084998)

[Sprite Images 4](#_Toc307084999)

[Palettes 4](#_Toc307085000)

[FrameBuffers 5](#_Toc307085001)

# Available Resolutions

The following video resolutions are available:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | FrameBuffer | | | | | BG | |
| Resolution | RGB | 256 | 16 | 4 | 2 | BG0,1 | BG2,3 |
| 800 x 600 @ 56Hz | N | N | N | N | N | 16 | N |
| 640 x 480 @ 87Hz | N | N | N | N | N | 16 | N |
| 400 x 300 @ 56Hz | N | N | N | Y | Y | 16/256 | 16/256 |
| 320 x 240 @ 87Hz | N | N | N | Y | Y | 16/256 | 16/256 |
| 200 x 150 @ 56Hz | N | Y | Y | Y | Y | 16/256 | 16/256 |
| 160 x 120 @ 87Hz | N | Y | Y | Y | Y | 16/256 | 16/256 |

# Video Modes

## RGB

At 400 x 300, 2 bytes per pixel, it would cost about 234K to store an image.

At 160 x 120, 2 bytes per pixel, it would cost about 37K to store an image.

Since there’s only 32K of VRAM, this is not an option in any resolution.

## 256 Color

At 400 x 300, 1 byte per pixel, it would cost about 117K to store an image.

At 320 x 240, 1 byte per pixel, it would cost about 75K to store an image.

At 200 x 150, 1 byte per pixel, it would cost about 29K to store an image.

At 160 x 120, 1 byte per pixel, it would cost about 18K to store an image.

Therefore, in low resolutions, it should be possible to support 256 color mode.

Fetch AAAAAAAABBBBBBBB

A and B are each a palette entry for two sequential pixels.

## 16 Color

At 400 x 300, 2 pixels per byte, it would cost about 58K to store an image.

At 320 x 240, 2 pixels per byte, it would cost about 37K to store an image.

At 200 x 150, 2 pixels per byte, it would cost about 14K to store an image.

At 160 x 120, 2 pixels per byte, it would cost about 9K to store an image.

Therefore, in low resolutions, it should be possible to support 16 color mode.

## 4 Color

At 400 x 300, 4 pixels per byte, it would cost about 29K to store an image.

At 320 x 240, 4 pixels per byte, it would cost about 18.5K to store an image.

At 200 x 150, 4 pixels per byte, it would cost about 7K to store an image.

At 160 x 120, 4 pixels per byte, it would cost about 4.5K to store an image.

Therefore, all non-native resolutions, should be able to support 4 color mode.

## 2 Color

At 800 x 600, 8 pixels per byte, it would cost about 60K to store an image.

At 640 x 480, 8 pixels per byte, it would cost about 38K to store an image.

At 400 x 600, 8 pixels per byte, it would cost about 30K to store an image.

At 400 x 300, 8 pixels per byte, it would cost about 14.5K to store an image.

At 320 x 240, 8 pixels per byte, it would cost about 9.25K to store an image.

At 200 x 150, 8 pixels per byte, it would cost about 3.5K to store an image.

At 160 x 120, 8 pixels per byte, it would cost about 2.25K to store an image.

Therefore, all non-native resolutions, should be able to support 2 color mode.

Fetch AAAABBBBCCCCDDDD

A, B, C, and D are each a palette entry for 4 sequential pixels.

## Display Layers

There are 4 hardware display layers. Each layer can display image data in the form of a framebuffer, or using 8x8 character cells. Sprites are completely independent of the mode.

Layer Registers

* Enabled = yes/no.
* Available = yes/no (internally determined by looking at current resolution)
* Mode (0 = Framebuffer, 1 = Character)
* BitDepth (0 = 4bit, 1 = 8bit)
* PaletteAddress = Address is 0XXXXXXX XXX00000 (32K @ 32byte granularity)
* ImageAddress (see each mode for specifics)

### Character Mode

Fetch CCCCCCCC

C = Character Index

Character Indexes are now simplified. No palette#, no flipping, because the extra bits were causing a single BG map to cost way too much VRAM. Even with just 8bits, a 128x128 map costs 16K of VRAM. A 64x64 map costs 4K of VRAM, and 4 BG’s cost a total of 16K, with no cell or sprite images present yet!

BG Control Registers (4 BG’s are available)

* Width = 128, 64, 32 or 16 tiles.
* Height = 128, 64, 32 or 16 tiles.
* MapAddress = Address is 0XXXXXX0 00000000 (32K @ 512byte granularity)
* ImageAddress (aka TilesetAddress) = Address is 0XXXXXX0 00000000 (32K @ 512byte granularity)

### FrameBuffer Mode

In framebuffer mode, a display is rendered using bytes from VRAM as palette indexes. There is an internal address of the next pixel that is always updating. At the top of the frame, it takes the AddressOfFirstPixel. At the end of each scanline, the modulo is added to it. With some tricky coding, you can create unusual video effects with a system like this.

The following registers are available:

Framebuffer Registers Available (4 framebuffers are available):

* ImageAddress (aka AddressOfFirstPixel) = 0XXXXXXX XXXXXX00 (32K @ 4byte granularity)
* Modulo = offset from end of line to start of next line 0XXXXXXX XXXXXX00 (32K @ 4byte granularity)

# VRAM

32KB VRAM available. It is the 32K of internal, super fast dual ported RAM inside the Xilinx chip.

Refer to the VRAM Map in the Graphics Spreadsheet, to see how this VRAM can be arranged into useful configurations.

## Kinds of VRAM

VRAM can be assigned to several different video-related features.

* Character Maps
* Character Cells
* Sprite Images
* Palettes
* FrameBuffers

Any indexed color mode (256 colors, 16 colors, 4 colors, or 2 colors) requires the use of a palette. RGB color modes are not supported, because even in the lowest resolution, 16bit RGB data requires way more RAM than the Xilinx has available. 8bit RGB could fit, but what’s the point, you could use 8bit indexed instead, for your choice of colors through the use of a palette.

### Character Maps

A character map is an array of indexes into a block of character cells. Each index is 8 bits.

### Character Cells

A character cell is an 8x8 block of pixels.

### Sprite Images

A sprite is a string of 4 u32’s wide per scanline which results in 32 pixels wide (4bit), or 16 pixels wide (8bit). The next 4 u32’s create another scanline. A sprite can be up to 64 pixels tall.

It is possible to link multiple sprites together. A sprite with the LINK bit set, ignores its’ own coordinates and glues itself to the immediate right of the previous sprite.

### Palettes

A palette can be assigned to any VRAM address that is aligned to 512 bytes.

Therefore, it requires the use of 6 bits (0..63 -> 0.. 32256) to describe its’ location in VRAM.

The length of the palette is implied by the bit-depth of the image that is making use of the palette. Eg. You could have some 4bit sprites and some 8bit sprites, and point each of them to different palettes, or share palettes. It’s up to you.

### FrameBuffers

In framebuffer mode, a display is rendered using bytes from VRAM as palette indexes.